# A Supplementary Materials

For easy reference we have summarized the basic algorithm in Pseudocode 1.

**function** $\mathcal{O} \leftarrow$ Synthesize($\mathcal{I}$)
  // $\mathcal{O}$: *output*
  // $\mathcal{I}$: *input exemplar*
  $\mathcal{O}_0 \leftarrow$ Initialize($\mathcal{I}$)
  **foreach** time step $t$
    $\mathcal{O}_t \leftarrow$ Advect($\mathcal{O}_{t-1}$) // *application specific advection*
    $\mathcal{O}_t \leftarrow$ Optimize($\mathcal{I}$, $\mathcal{O}_t$) // *compute one frame*
  **end**

**function** $\mathcal{O}_t \leftarrow$ Optimize($\mathcal{I}$, $\mathcal{O}_t$)
  **iterate** until convergence **or** enough # of iterations reached
    $\{\mathbf{n}(s_i)\} \leftarrow$ Search($\mathcal{O}_t$, $\mathcal{I}$) // *search phase*
    Assign($\{\mathbf{n}(s_i)\}$, $\mathcal{O}_t$) // *assignment phase*
    extra solver steps
  **end**
  **return** $\mathcal{O}_t$

**function** $\{\mathbf{n}(s_i)\} \leftarrow$ Search($\mathcal{O}_t$, $\mathcal{I}$)
  **foreach** element $s_o \in \mathcal{O}_t$
    $\mathbf{n}(s_o) \leftarrow$ output neighborhood around $s_o$
    $\mathbf{n}(s_i) \leftarrow$ find most similar neighborhood to $\mathbf{n}(s_o)$ for $s_i \in \mathcal{I}$
  **end**
  **return** $\{\mathbf{n}(s_i)\}$

**function** Assign($\{\mathbf{n}(s_i)\}$, $\mathcal{O}_t$)
  **foreach** output element $s_o \in \mathcal{O}_t$
    $\mathbf{p}(s_o) \leftarrow$ least squares from predicted positions
  **end**

**Pseudocode 1:** *Dynamic element textures for frame-by-frame synthesis.*

## A.1 Algorithm Enhancements

Here we describe enhancements in speed and quality of our basic algorithm.

**Factorization** Due to the high dimensionality of the state spaces for dynamic element textures, we cannot expect small input exemplars to provide sufficient coverage for all plausible configurations. Fortunately, we have observed that many phenomena have only loosely coupled spatial geometries and temporal motions. This allows us to perform factorization to enhance synthesis quality. Specifically, given an output neighborhood, we first extract its spatial and temporal parts (the horizontal and vertical portions as illustrated in Figure 3 right), and find the corresponding spatial- and temporal-only best matches in the search step. Both of the two matches provide its own predictions for the relative positions during the assignment step.

**Order-independence** Our method can be made order-independent [Lefebvre and Hoppe 2005] in addition to being parallel. That is, it can compute a specific spatial-temporal subset of the output without touching the entire volume, with consistent results regardless of the chosen subset. The key idea, similar to prior order-independent methods, is to gradually expand the footprint from the output subset towards earlier iterations, and synthesize the footprint from earlier to later iterations. Order-independence can be very helpful in a variety of scenarios, such as parallel computing multiple frames of the same animation, or interactive editing for a specific spatial-temporal constraint.

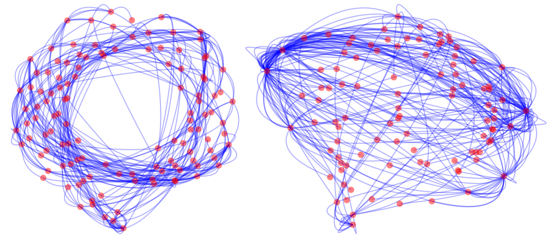

**Figure 15:** *Neighborhood graph examples. Here we visualize the neighborhood graph by projecting high dimensional neighborhoods onto 2D plane via principal component analysis. Each red point represents an input spatial neighborhood while the blues lines represent generated motion paths between similar neighborhoods. See the tree results in the accompanying video for the corresponding animations.*

## A.2 Usage

To use our system, users simply specify the detailed input exemplar and coarse output constraints.

**Input exemplar** Our method is agnostic with respect to how the input exemplars are prepared; potential methods include manual specification, procedural modeling + animation, physical simulation, and data capture. Note that once an input exemplar is prepared, it can be reused for many different outputs.

**Output constraint** Output constraints for our system can be classified along two main orthogonal dimensions: local versus global $\times$ spatial and/or temporal. In particular, a local/global constraint affects a small/large set of samples, and a spatial/temporal constraint is about static-shapes/dynamic-motions. Our system simply treats all constraints as collections of individual spatial-temporal sample constraints and feed them into our constrained synthesis algorithm in Section 4.1.

There are also additional constraints, such as physical sub-solver, that do not neatly fit into the classification above. These can be specified as additional energy terms or solver steps as described in Section 4.1.